Building Hardware Systems for Information Flow Tracking

Hari Kannan

Computer Systems Laboratory

Stanford University

The Computer Security Crisis

- More systems are online, vulnerable
  - Banking, Power, Water, Government
- Threats have multiplied
  - XSS, SQL Injection, Phishing, …
- Old challenges remain
  - Buffer overflows, broken access control

A Blast from the Past?

The New York Times
nytimes.com

October 23, 2008

Attack code for critical Microsoft bug surfaces

By ROBERT MCMILLAN, IDG

Just hours after Microsoft posted details of a critical Windows bug, new attack code that exploits the flaw has surfaced.

It took developers of the Immunity security testing tool two hours to write their exploit, after Microsoft released a patch for the issue Thursday morning. Software developed by Immunity is made available only to paying customers, which means that not everyone has access to the new attack, but security experts expect that some version of the code will begin circulating in public very soon.

Microsoft took the unusual step of rushing out an emergency patch for the flaw Thursday, two weeks after noticing a small number of targeted attacks that exploited the bug.

The vulnerability was not publicly known before Thursday; however, by issuing its patch, Microsoft has given hackers and security researchers enough information to develop their own attack code.

The flaw lies in the Windows Server service, used to connect different network resources such as file and print servers over a network. By sending malicious messages to a Windows machine that uses Windows Server, an attacker could take control of the computer, Microsoft said.

Apparently, it doesn't take much effort to write this type of attack code.

"It is very exploitable," said Immunity Security Researcher Bas Albert. "It's a very controllable stack overflow."

Stack overflow bugs are caused when a programming error allows the attacker to write a command on parts of the computer's memory that would normally be out of limits and then cause that command to be run by the victim's computer.

Wave of the Future?

October 13th, 2008

The Image Group Website Hacked Through SQL-Injection, Credit Cards Data Stolen

From January to August 2008, hackers through an SQL injection flaw were able to access names and credit or debit card information of the persons who placed orders on The Image Group e-commerce website. The Image Group (http://www.theimagegroup.net) is a firm for promotional products and corporate merchandise headquartered in Ohio.

The Image Group has notified the New Hampshire State Attorney General and online customers that their e-commerce site fell victim to a series of successful SQL injection attacks. The compromised database contained sensitive personal and financial information belonging to customers of the company.

While this was discovered in August, it appears that the unauthorized access began in January and occurred again in August of this year. Names, credit cards/debit cards numbers, expiration dates, addresses and the CVV codes were accessed by hackers. No social security numbers or dates of birth were involved.

Upon learning of the breach, the firm shut down the web site through which the unauthorized access occurred. In addition, they had a forensic audit performed. Currently they are working with the merchant bank and the Card Associations to address issues associated with the credit card information taken and to notify the issuing banks for those cards.

It seems that the website domain related to this incident is theideacatalog.com, for which the registrant and administrative contact is Target Marketing. The IP address for the www.ideacatalog.com website is 74.84.205.104 and belongs to the block 74.84.205.0 - 74.84.205.255, which is assigned to Target Marketing. The site may have been developed and/or managed by Target Marketing.

Toll-free number for questions is 866-272-5162.

Source: cyberinsecure.com

Motivation

- Security research
  - Provide simple & practical abstractions for expressing and enforcing security policies
- The resulting system must be
  - Robust: protects against wide range of threats
  - Flexible: can be adjusted for future threats
  - Practical: works with all types of existing SW
  - End-to-end: protects both user and kernelspace code
  - Fast: no significant runtime overheads

Why Hardware Support?

- Advantages of HW support
  - Better performance
  - Fine-granularity protection
  - Lowest level of the system stack
    - Difficult to bypass, can build upon its guarantees
  - Simplify the SW security framework

- Our focus: combine the best of HW + SW
  - HW: low-level operations and enforcement
  - SW: high-level policies and analysis

DIFT: Dynamic Information Flow Tracking

- DIFT taints data from untrusted sources
  - Extra tag bit per word marks if untrusted
- Propagate taint during program execution
  - Operations with tainted data produce tainted results
- Check for unsafe uses of tainted data
  - Tainted code execution
  - Tainted pointer dereference (code & data)
  - Tainted SQL command
- Can detect both low-level & high-level threats

Thesis Overview

- Design practical hardware systems implementing Dynamic Information Flow Tracking (DIFT) for software security

- Thesis contributions
  - Co-developed a **flexible** hardware design for **efficient, practical** DIFT on binaries
    - Including a **real** full-system prototype (HW+SW)
  - Developed hardware mechanisms for DIFT to allow for **practical, cost-effective** implementation
    - Implemented a DIFT coprocessor (**real** full-system prototype)
  - Developed a mechanism for safe DIFT on multi-threaded binaries
  - Leveraged DIFT mechanisms and co-developed a **flexible** hardware design for **information flow control**
    - Hardware directly enforces application security policies
    - Allows for significant reduction in size of OS' trusted computing base
    - Including a **real** full-system prototype (HW+SW)

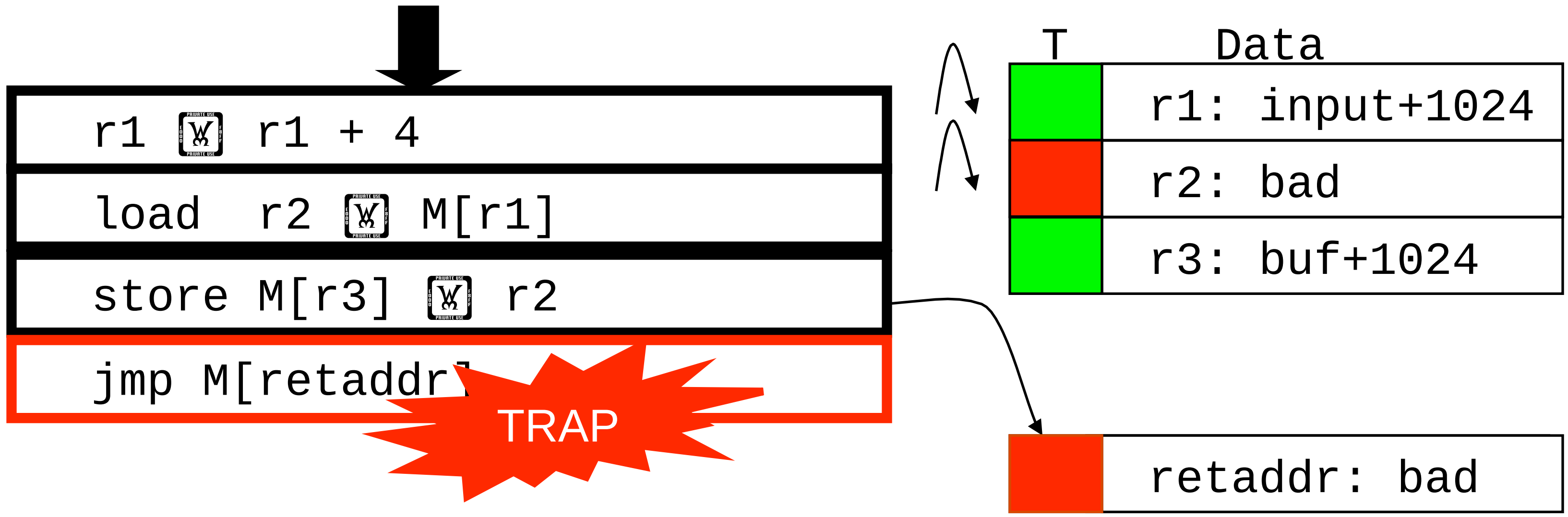
Outline

- DIFT overview
- Raksha: hardware support for DIFT [WDDD'06, ISCA'07]
  - Flexible HW design for efficient, practical DIFT on binaries
- Decoupling DIFT from the processor [DSN'09]
  - Using a coprocessor to minimize changes to the main core
- Multi-processor DIFT [MICRO'09]
  - Ensuring consistency between data and metadata under decoupling
- Loki: hardware support for information flow control [OSDI'08]
  - Enforcement of app security policies with minimal trusted code

DIFT Example: Memory Corruption

Vulnerable C Code

```
char buf[1024];
strcpy(buf,input);//buffer overflow
```



Tainted pointer dereference ➡ security trap

DIFT Example: SQL Injection

Username: christos' OR '1'='1
Password:

Vulnerable SQL Code

SELECT * FROM *table*

WHERE *name*= 'username';

WHERE *name*= 'christos' OR '1'='1' ;

T	Data
(untainted)	WHERE name=
(tainted)	christos
(tainted)	OR
(tainted)	1=1

Tainted SQL command ➲ security trap

Implementing DIFT on Binaries

- Software DIFT [Newsome'05, Quin'06]
  - Use Dynamic Binary Translation (DBT) to implement DIFT
  - ☑ Runs on existing hardware, flexible security policies
  - ☒ **High overheads** (3–40x), **incompatible** with threaded or self-modifying code, limited to a single core

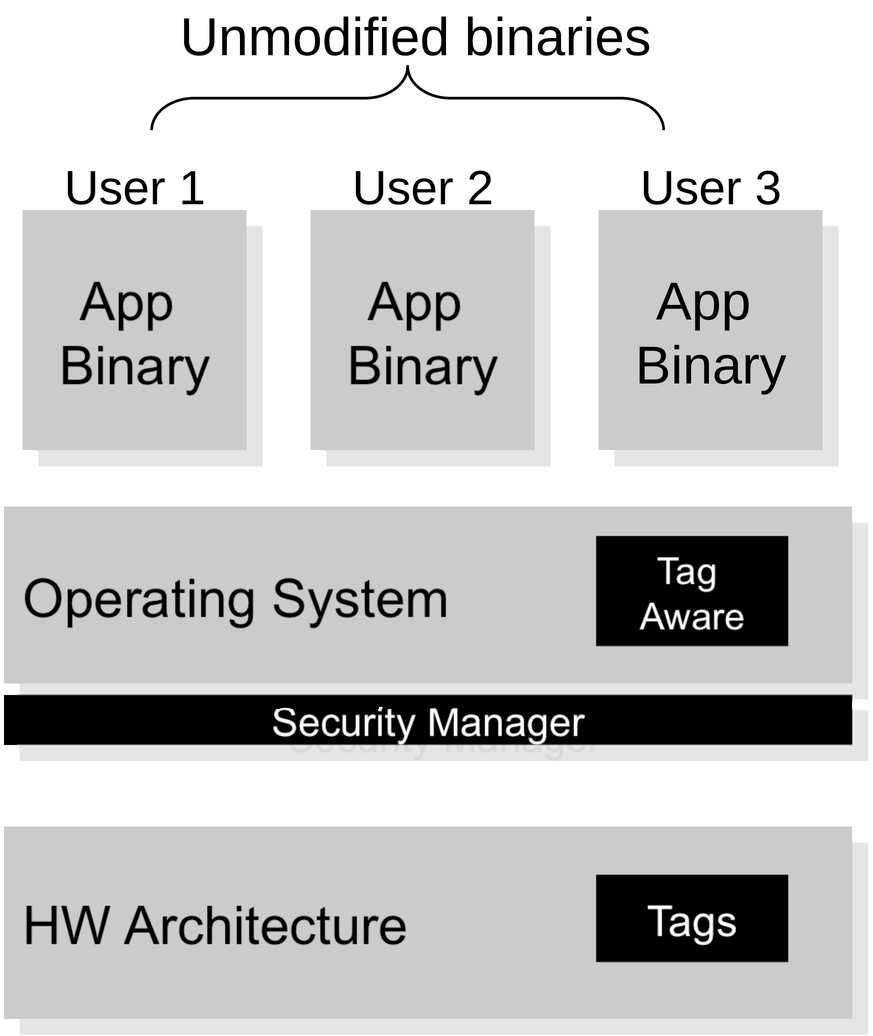
- Hardware DIFT [Suh'04, Crandall'04, Chen'05]
  - Modify CPU caches, registers, memory consistency, DRAM
  - ☑ Negligible overhead, works for all types of binaries, multi-core
  - ☒ **Inflexible** policies (false positives/negatives), cannot protect OS

- **Best of both worlds**
  - HW for tag propagation and checks
  - SW for policy management and high-level analysis
  - <u>Robust, flexible, practical, end-to-end, and fast</u>

Outline

- DIFT overview
- Raksha: hardware support for DIFT [WDDD'06, ISCA'07]
  - Flexible HW design for efficient, practical DIFT on binaries
- Decoupling DIFT from the processor [DSN'09]
  - Using a coprocessor to minimize changes to the main core
- Multi-processor DIFT [MICRO'09]
  - Ensuring consistency between data and metadata under decoupling
- Loki: hardware support for information flow control [OSDI'08]
  - Enforcement of app security policies with minimal trusted code

Raksha System Overview



- Save/restore tags
- Cross-process info flow

- Set HW security policies
- Further SW analysis

- 4 tag bits per word
- Programmable check/propagate
- User-level security traps

HW/SW Interface for DIFT Policies

- A pair of policy registers per tag bit
  - Set by security manager (SW) when and as needed
- Policy granularity: operation type
  - Select input operands to be checked for taint
  - Select input operands that propagate taint to output
  - Select the propagation mode (and, or, xor)
- ISA instructions decomposed to ≥1 operations
  - Types: ALU, comparison, insn fetch, data movement, …
  - Makes policies independent of ISA packaging
    - Same HW policies for both RISC & CISC ISAs
    - Don’t care how operations are packaged into ISA insns

Propagate Policy Example: `load`

`load r2 ← M[r1+offset]`

Propagate Enables

1. Propagate only from source register
   Tag(r2) ← Tag(r1)
2. Propagate only from source address
   Tag(r2) ← Tag(M[r1+offset])
3. Propagate only from both sources
   OR mode: Tag(r2) ← Tag(r1) | Tag(M[r1+offset])
   AND mode: Tag(r2) ← Tag(r1) & Tag(M[r1+offset])
   XOR mode: Tag(r2) ← Tag(r1) ^ Tag(M[r1+offset])

Check Policy Example: `load`

`load  r2 ← M[r1+offset]`

Check Enables

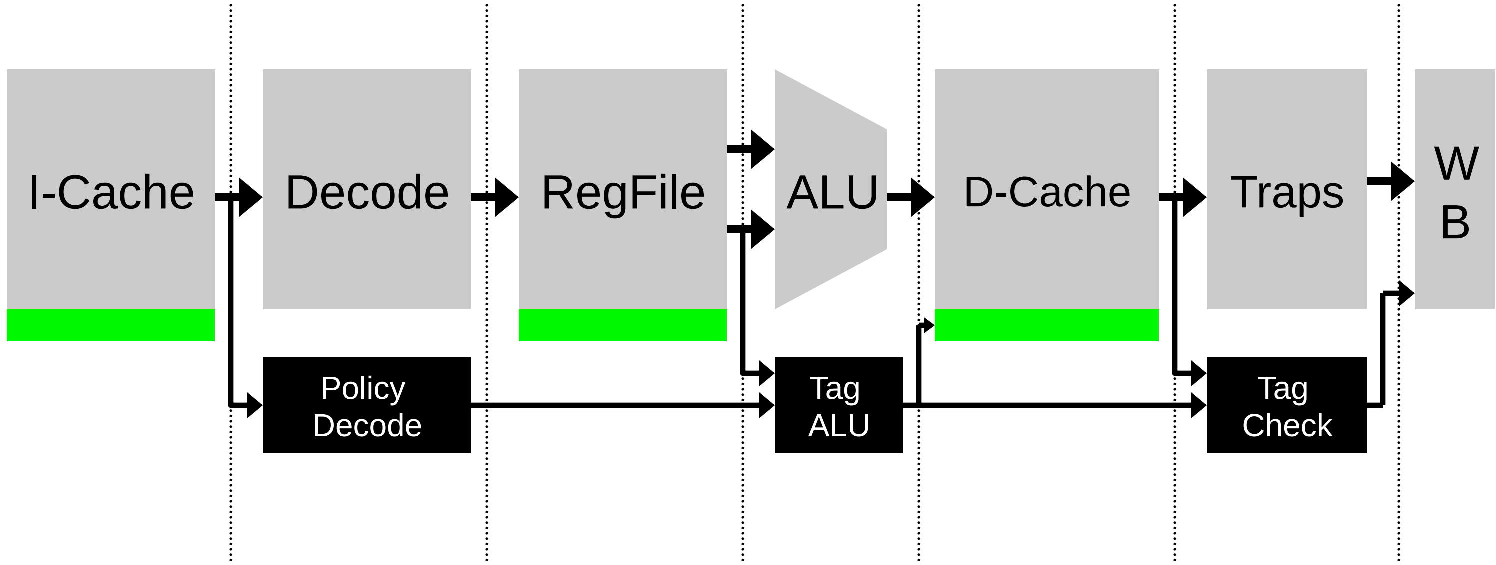
1. Check source register
   If Tag(`r1`)==1 then security_trap

2. Check source address
   If Tag(`M[r1+offset]`)==1 then security_trap

Both enables may be set simultaneously

Support for checks across multiple tag bits

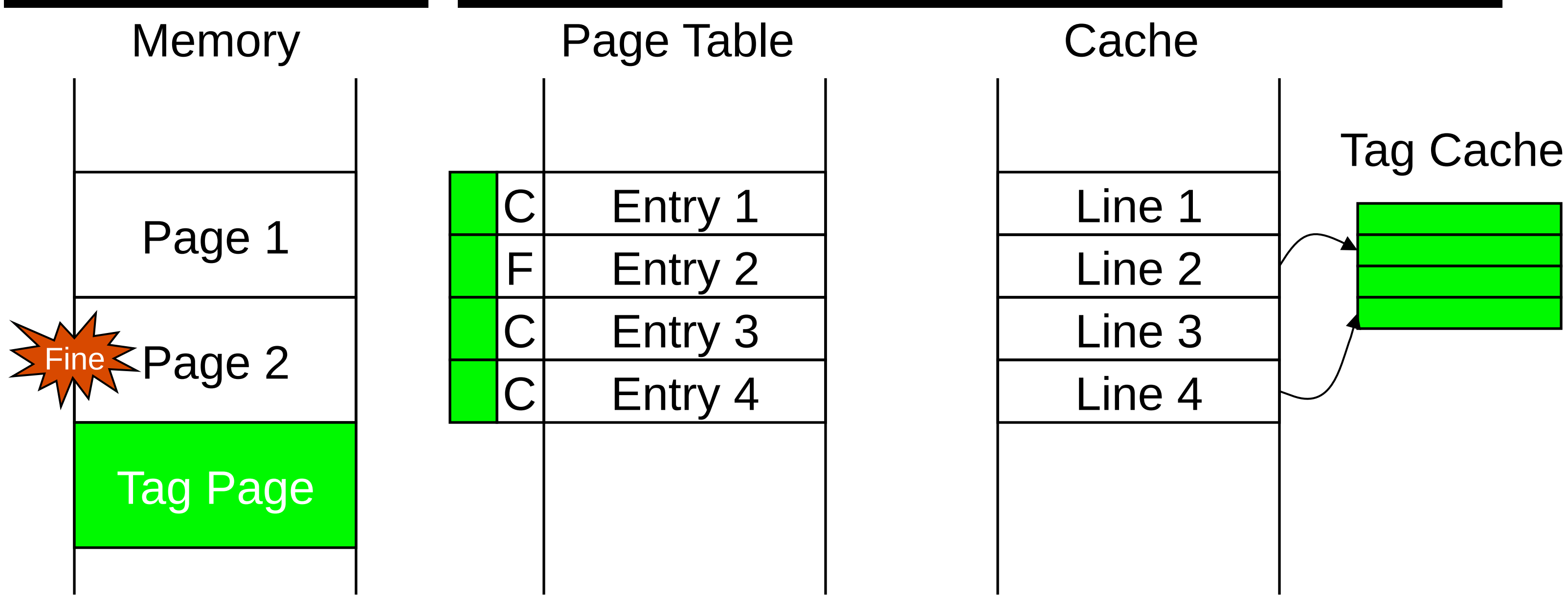
Raksha Hardware



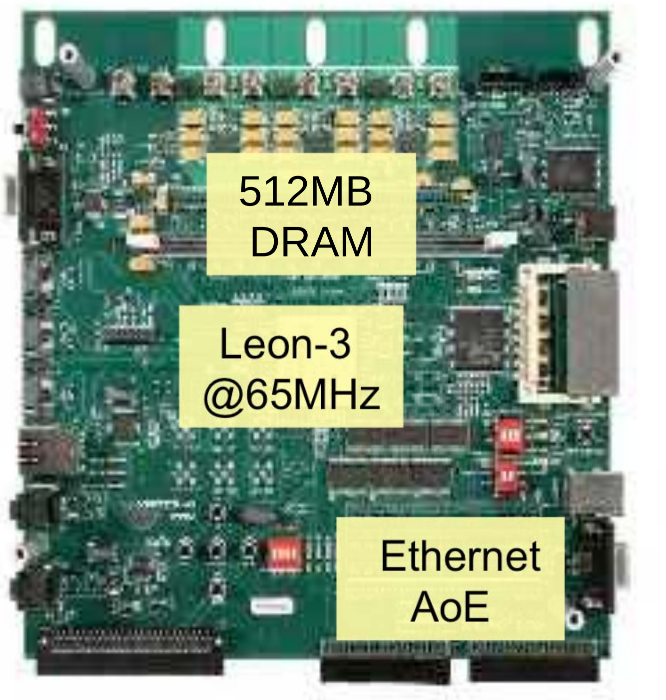
- Registers, caches & memory extended with tag bits
  - 4 tag bits per word of memory
- Tags flow through pipeline along with corresponding data
  - No changes in forwarding logic

Tag Storage

- Simple approach: +4 bits/word in registers, caches, memory
  - 12.5% storage overhead
  - Used in our original prototype
- Multi-granular tag storage scheme
  - Exploit tag locality to reduce storage overhead (~1-2%)
  - Page-level tags ➔ cache line-level tags ➔ word-level tags



Raksha Prototype



- Hardware
  - Modified SPARC V8 CPU (LEON-3)
  - Mapped to FPGA board
- Software
  - Full-featured Gentoo Linux workstation
  - Used with >14k packages (LAMP, etc)
- Design statistics
  - Clock frequency: same as original
  - Logic: +4.3% overhead
  - Performance: <1% slowdown
    - Across a wide range of applications
    - SW DIFT is 3-40x slowdown

Security Policies Overview

		P Bit	T Bit	B Bit	S Bit
Buffer Overflow Policy	Identify all pointers, and track data taint. Check for illegal tainted ptr use.	Y	Y		
Offset-based attacks (control ptr)	Track data taint, and bounds check to validate.			Y	
Format String Policy	Check tainted args to print commands.		Y		Y
SQL / XSS	Check tainted commands.		Y		Y
Red zone Policy	Sandbox heap data.				Y
Sandboxing Policy	Protect the security handler.				Y

Security Experiments

Program	Lang.	Attack	Detected Vulnerability
tar	C	Directory Traversal	Open tainted dir
gzip	C	Directory Traversal	Open tainted dir
Wu-FTPD	C	Format String	Tainted '%n' in vfprintf string
SUS	C	Format String	Tainted '%n' in syslog
quotactl syscall	C	User/kernel pointer dereference	Tainted pointer to kernelspace
sendmail	C	Buffer (BSS) Overflow	Tainted code ptr
polymorph	C	Buffer Overflow	Tainted code ptr
OpenSSH	C	Command Injection	Execve tainted file
ProFTPD	C	SQL Injection	Tainted SQL command
htdig	C++	Cross-site Scripting	Tainted <script> tag
Scry	PHP	Cross-site Scripting	Tainted <script> tag

- Unmodified SPARC binaries from real-world programs
  - Basic/net utilities, servers, web apps, search engine

Security Experiments

Program	Lang.	Attack	Detected Vulnerability
tar	C	Directory Traversal	Open tainted dir
gzip	C	Directory Traversal	Open tainted dir
Wu-FTPD	C	Format String	Tainted '%n' in vfprintf string
SUS	C	Format String	Tainted '%n' in syslog
quotactl syscall	C	User/kernel pointer dereference	Tainted pointer to kernelspace
sendmail	C	Buffer (BSS) Overflow	Tainted code ptr
polymorph	C	Buffer Overflow	Tainted code ptr
OpenSSH	C	Command Injection	Execve tainted file
ProFTPD	C	SQL Injection	Tainted SQL command
htdig	C++	Cross-site Scripting	Tainted <script> tag
Scry	PHP	Cross-site Scripting	Tainted <script> tag

- Protection is independent of programming language
  - Propagation & checks at the level of basic ops

Security Experiments

Program	Lang.	Attack	Detected Vulnerability
tar	C	Directory Traversal	Open tainted dir
gzip	C	Directory Traversal	Open tainted dir
Wu-FTPD	C	Format String	Tainted '%n' in vfprintf string
SUS	C	Format String	Tainted '%n' in syslog
quotactl syscall	C	User/kernel pointer dereference	Tainted pointer to kernelspace
sendmail	C	Buffer (BSS) Overflow	Tainted code ptr
polymorph	C	Buffer Overflow	Tainted code ptr
OpenSSH	C	Command Injection	Execve tainted file
ProFTPD	C	SQL Injection	Tainted SQL command
htdig	C++	Cross-site Scripting	Tainted <script> tag
Scry	PHP	Cross-site Scripting	Tainted <script> tag

- Protection against low-level memory corruptions
  - Both control & non-control data attacks

Security Experiments

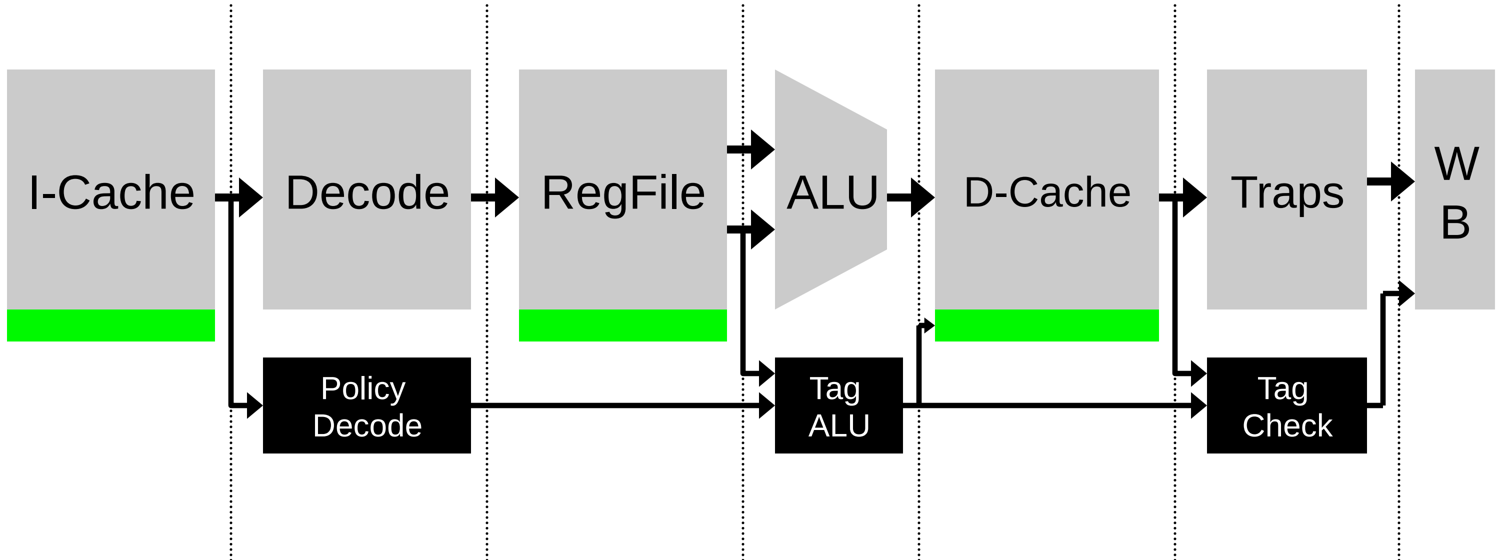
Program	Lang.	Attack	Detected Vulnerability
tar	C	Directory Traversal	Open tainted dir
gzip	C	Directory Traversal	Open tainted dir
Wu-FTPD	C	Format String	Tainted ‘%n’ in vfprintf string
SUS	C	Format String	Tainted ‘%n’ in syslog
quotactl syscall	C	User/kernel pointer dereference	Tainted pointer to kernelspace
sendmail	C	Buffer (BSS) Overflow	Tainted code ptr
polymorph	C	Buffer Overflow	Tainted code ptr
OpenSSH	C	Command Injection	Execve tainted file
ProFTPD	C	SQL Injection	Tainted SQL command
htdig	C++	Cross-site Scripting	Tainted <script> tag
Scry	PHP	Cross-site Scripting	Tainted <script> tag

- 1st hardware DIFT system to detect high-level attacks
  - No false positives observed

Outline

- DIFT overview
- Raksha: hardware support for DIFT [WDDD'06, ISCA'07]
  - Flexible HW design for efficient, practical DIFT on binaries
- Decoupling DIFT from the processor [DSN'09]
  - Using a coprocessor to minimize changes to the main core
- Multi-processor DIFT [MICRO'09]
  - Ensuring consistency between data and metadata under decoupling
- Loki: hardware support for information flow control [OSDI'08]
  - Enforcement of app security policies with minimal trusted code

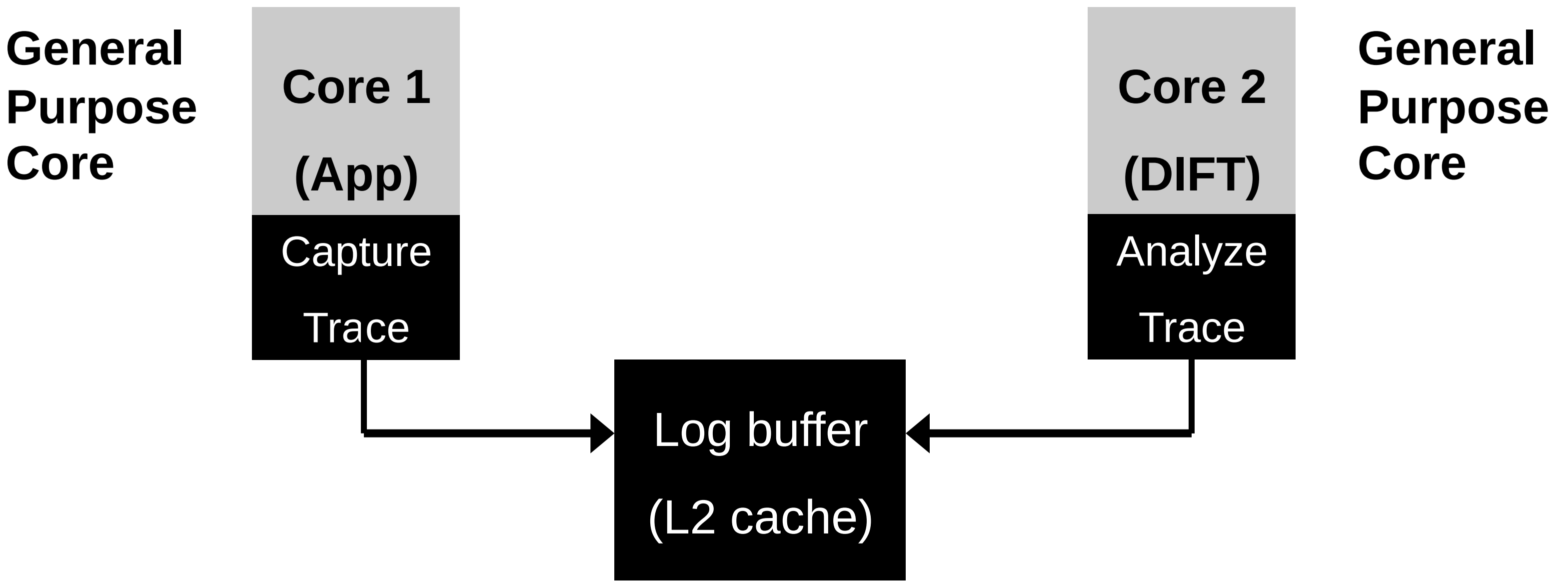
HW Option 1: In-core DIFT



- Integrated DIFT hardware [Dalton'07, Suh'04, Chen'05]
  - No performance, minor power, and minor area overhead
  - Invasive changes to processor
  - High design and validation costs

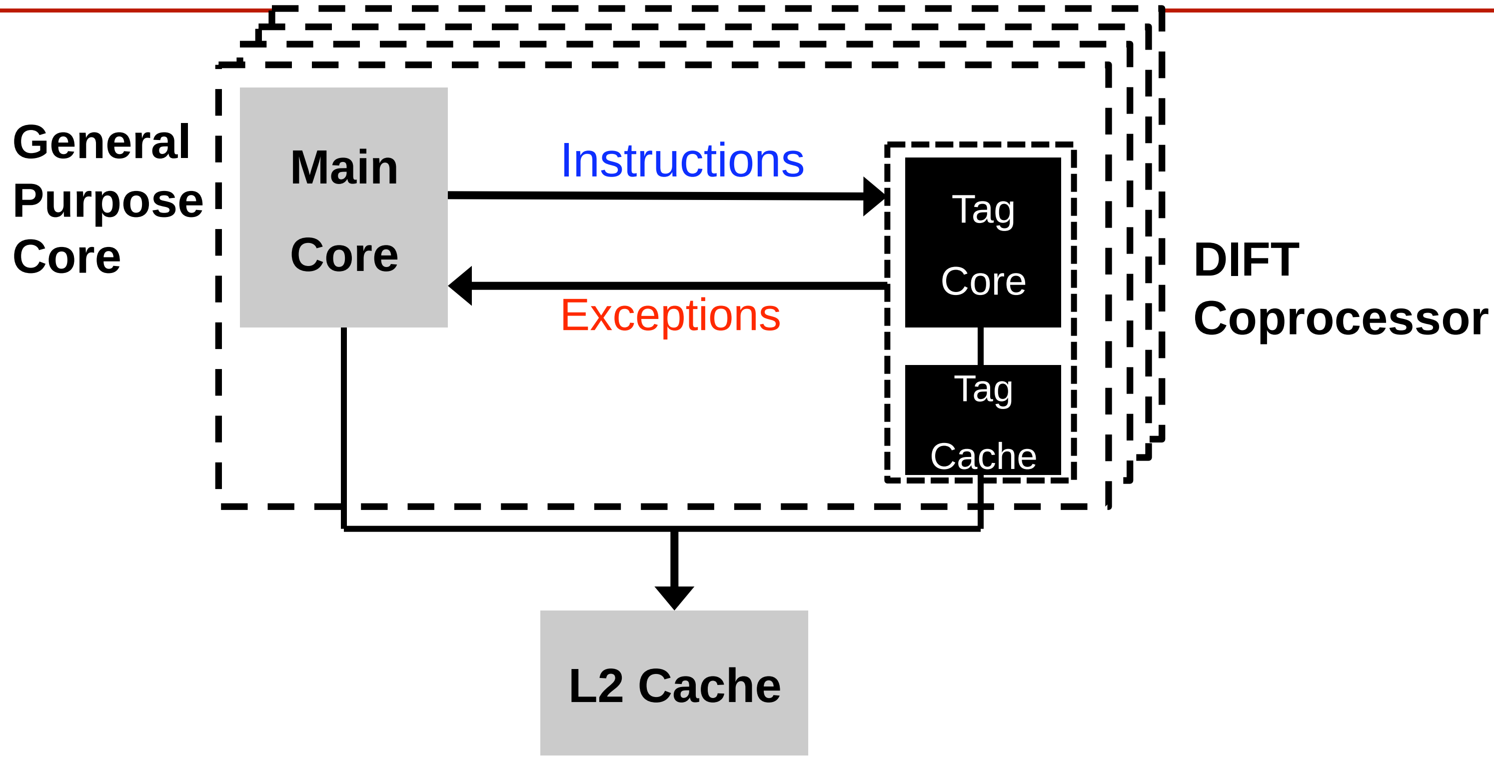
- Synchronizes metadata and data per instruction

HW Option 2: Offloading DIFT



- **SW DIFT** on modified multi-core chip (e.g., CMU's LBA)
  - Flexible support for various analyses
  - Large area & power overhead (2nd core, trace compress)
  - Large performance overhead (DBT, memory traffic)
  - Significant changes to processor & memory hierarchy

Our Proposal: DIFT Coprocessor



- Off-core DIFT coprocessor (similar to watchdog processors)
  - Small performance, power, and area overhead
  - Minor changes to processor
  - Reuse across processor designs

What happens without Proc/Coproc Synchronization?

Vulnerable C Code

```
int idx = tainted_input;
buffer[idx] = x; // memory corruption
```

```
set   r1 ←&tainted_input
load  r2 ← M[r1]
add   r4 ← r2 + r3
store M[r4]
…
exec (sys call)
```

SYSTEM COMPROMISE

T	Data
	r1:&input
	r2:idx=input
	r3:&buffer
	r4:&buffer+idx
	r5:x

- Attacker executes system call ➲ **system compromise**

System Calls as Sync points

- **Key Idea:** Main core and coproc sync at system calls

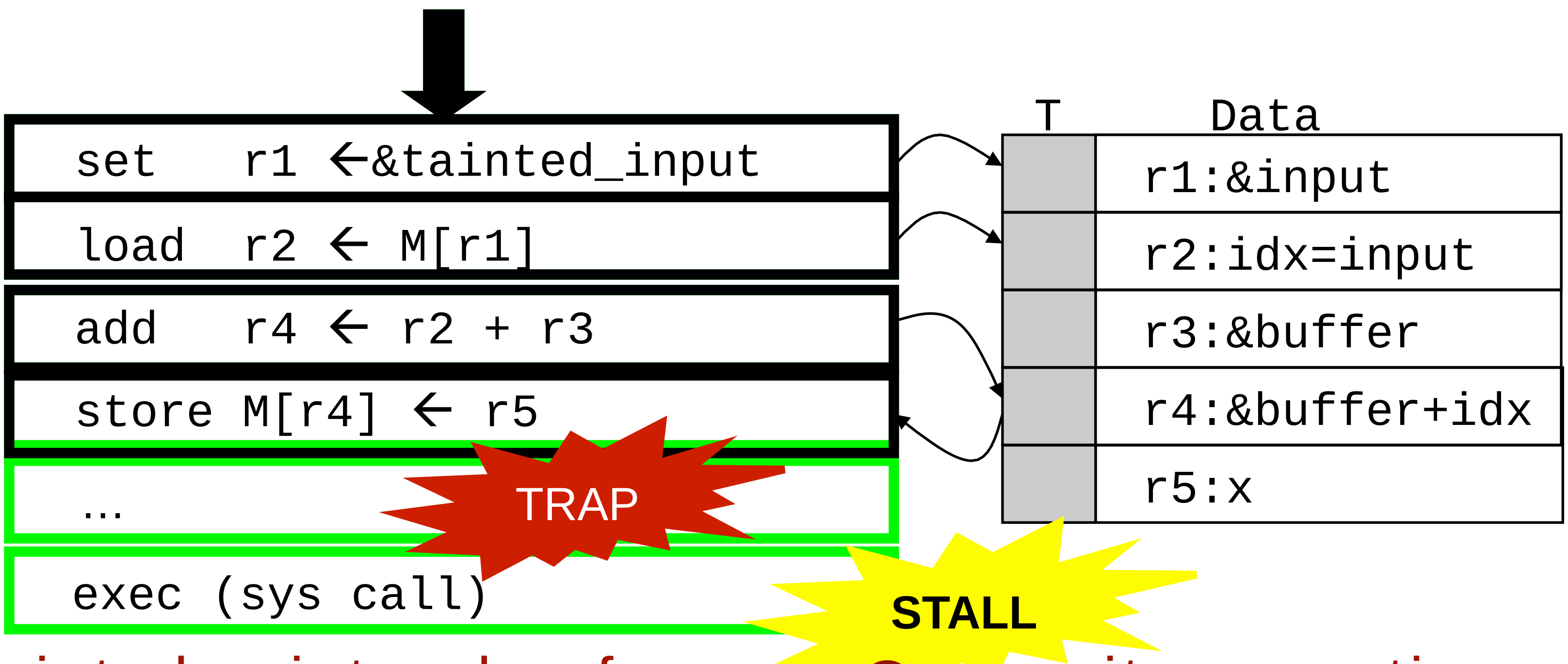
- Security:
  - This prevents attacker from executing system calls
  - Application's corrupted address space can be discarded
  - Does not weaken the DIFT model
    - DIFT detects attack only at time of **exploit**, not corruption

- Performance:
  - Synchronization overhead typically tens of cycles
    - Function of decoupling queue size
  - Lost in the noise of system call overheads (hundreds of cycles)

System Call Synchronization

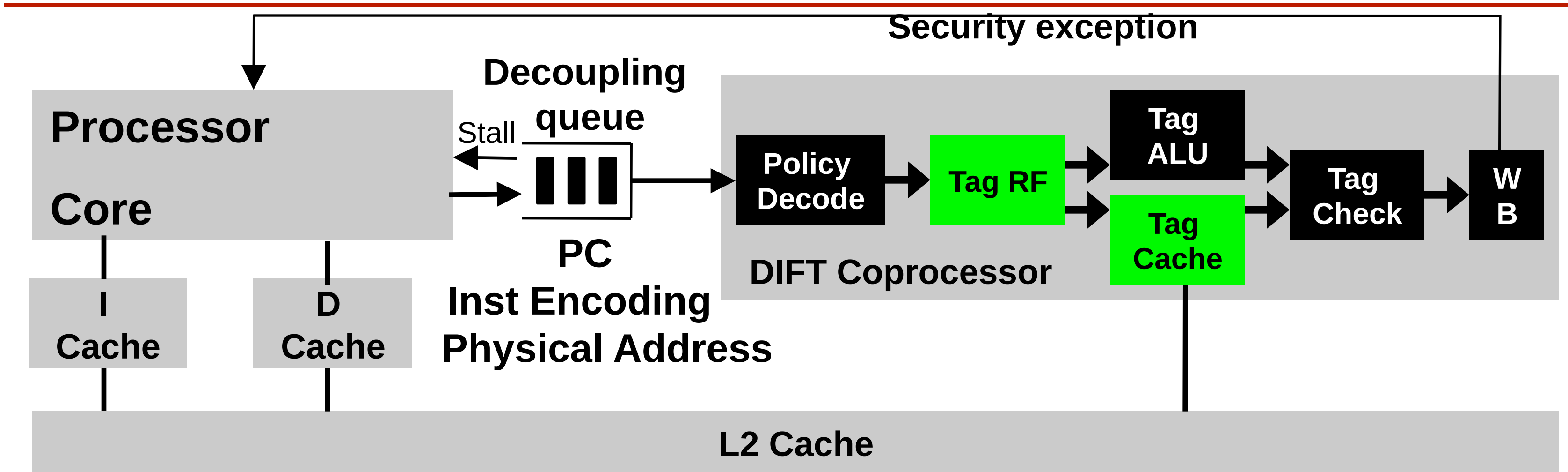
Vulnerable C Code

```
int idx = tainted_input;
buffer[idx] = x; // memory corruption
```



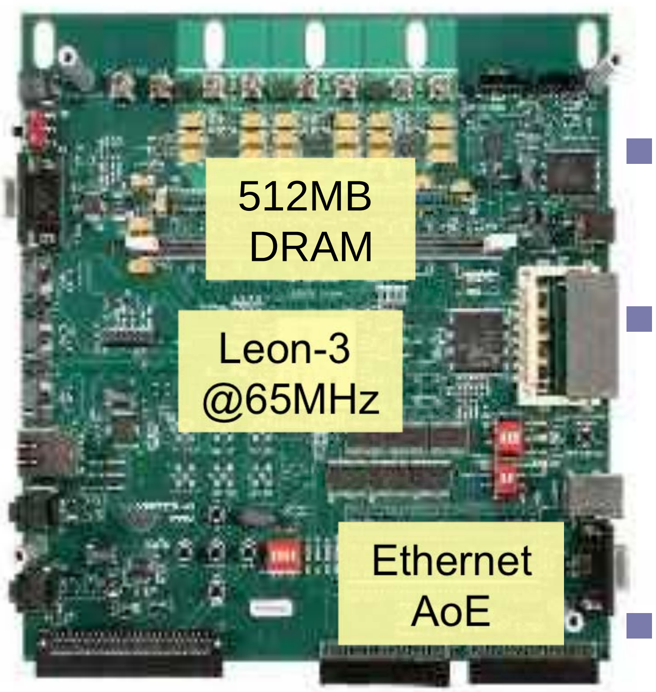
- Tainted pointer dereference ➡ security exception

Coprocessor Design



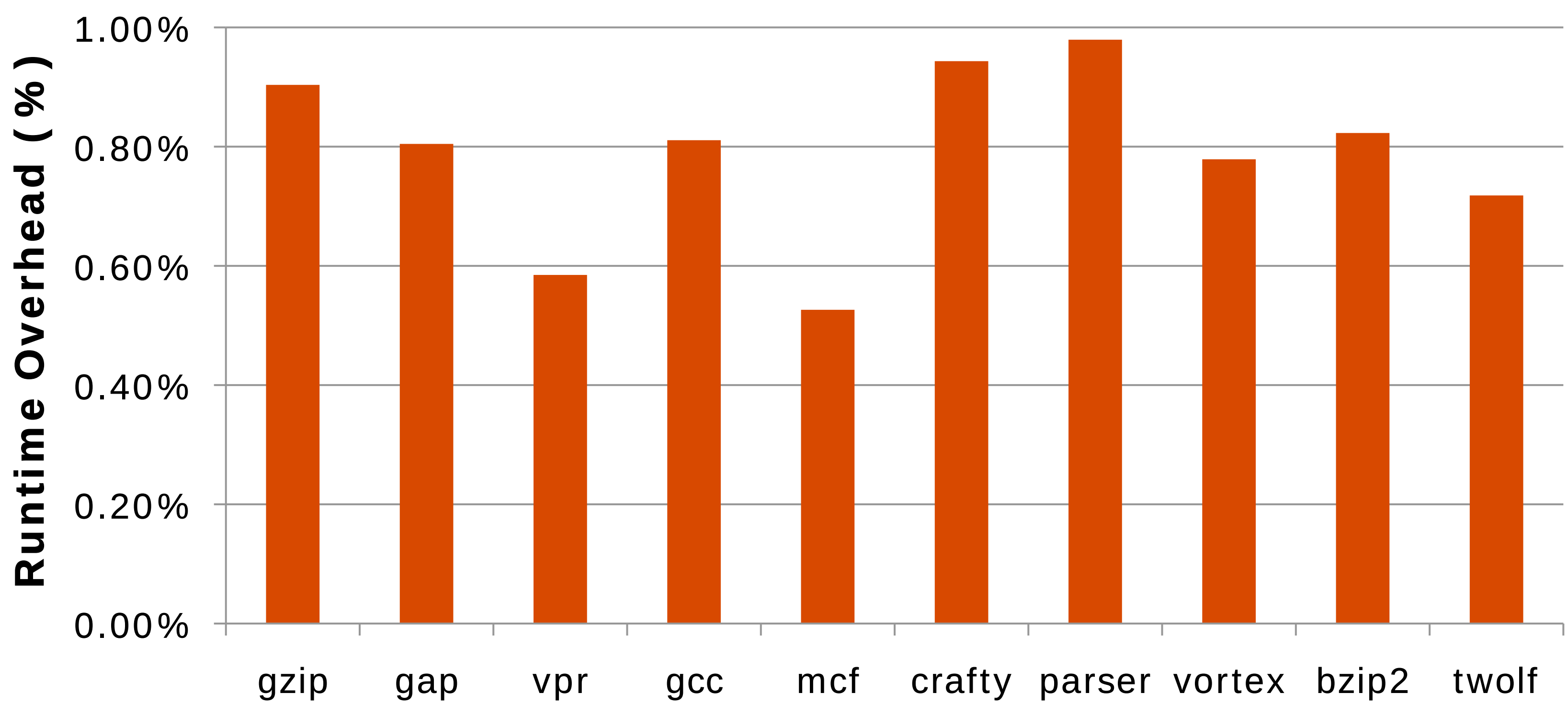
- DIFT functionality in a coprocessor
  - 4 tag bits of metadata per word of data
- Coprocessor Interface (via decoupling queue)
  - Pass committed instruction information
  - Instruction encoding could be at micro-op granularity (in x86)
  - Physical address obviates need for MMU in coprocessor

Prototype



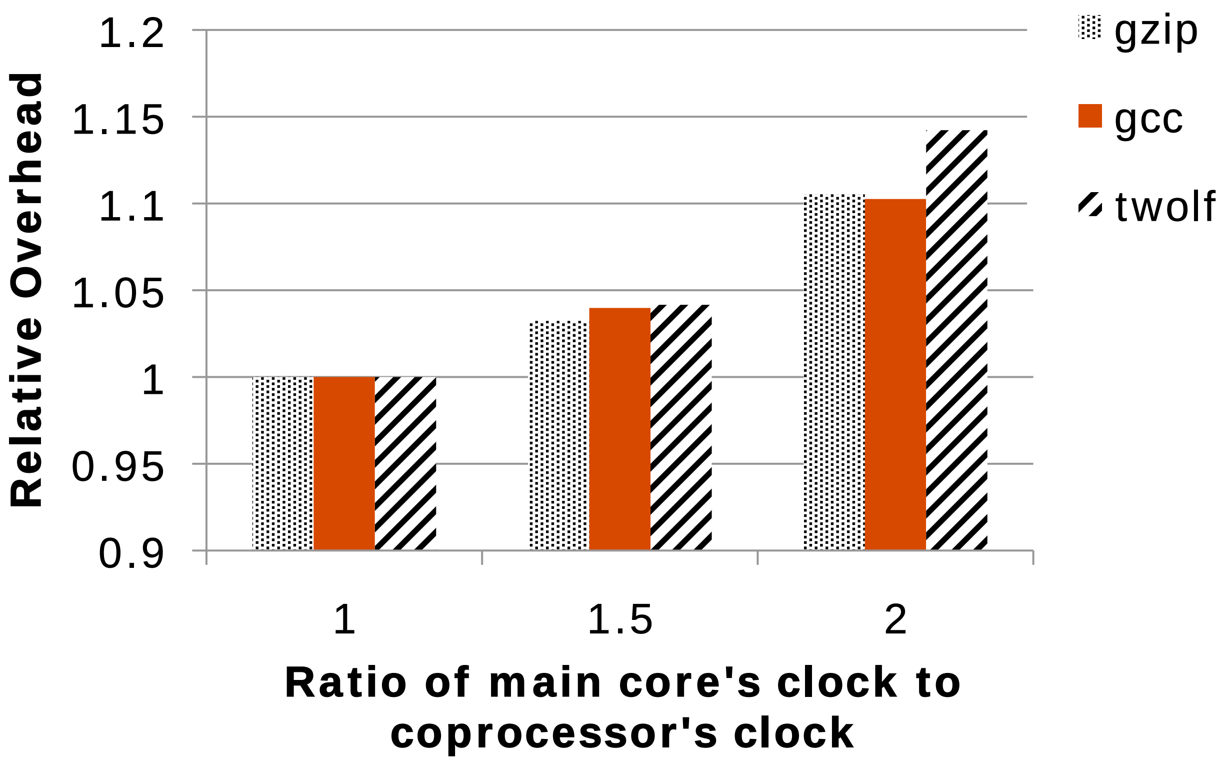
- Hardware
  - Paired with simple SPARC V8 core (Leon-3)
  - Mapped to FPGA board
- Software
  - Fully-featured Linux 2.6
- Design statistics
  - Clock frequency: same as original
  - Logic: +7.5% overhead
    - … of simple in-order core with no speculation
- Security
  - Catches same attacks as Raksha
  - No false positives or negatives

System Performance Overheads



- Runtime overhead **< 1%** over SPEC benchmarks
  - 512 byte tag cache
  - 6-entry decoupling queue

Coprocessors for complex cores



- Modest overheads with higher IPC cores
  - Because main core rarely achieves peak IPC (=1)
  - Coprocessor performs very simple operations
- Implies coprocessor can be paired with complex cores

Outline

- DIFT overview
- Raksha: hardware support for DIFT [WDDD'06, ISCA'07]
  - Flexible HW design for efficient, practical DIFT on binaries
- Decoupling DIFT from the processor [DSN'09]
  - Using a coprocessor to minimize changes to the main core
- Multi-processor DIFT [MICRO'09]
  - Ensuring consistency between data and metadata under decoupling
- Loki: hardware support for information flow control [OSDI'08]
  - Enforcement of app security policies with minimal trusted code

The Consistency Problem

- Decoupling metadata breaks atomicity between data/tags
  - Leads to consistency issues in multiprocessors

Proc 1 | Proc 2 | Tag Proc 1 | Tag Proc 2

1. ***u*** = ***t*** (Proc 1)
2. ***x*** = ***u*** (Proc 2)
3. tag(***x***) = tag(***u***) (Tag Proc 2)
4. tag(***u***) = tag(***t***) (Tag Proc 1)

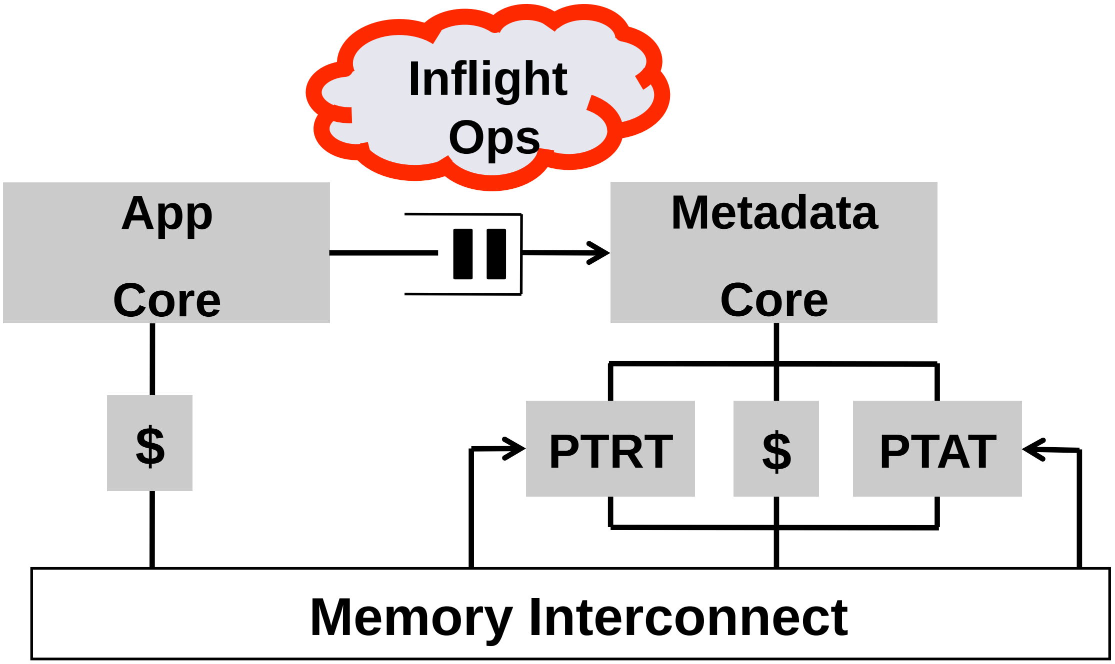
Inconsistency between data and metadata (x updated first)

- Can cause false positives/negatives
  - Spurious detections/miss real attacks

Fundamental Idea

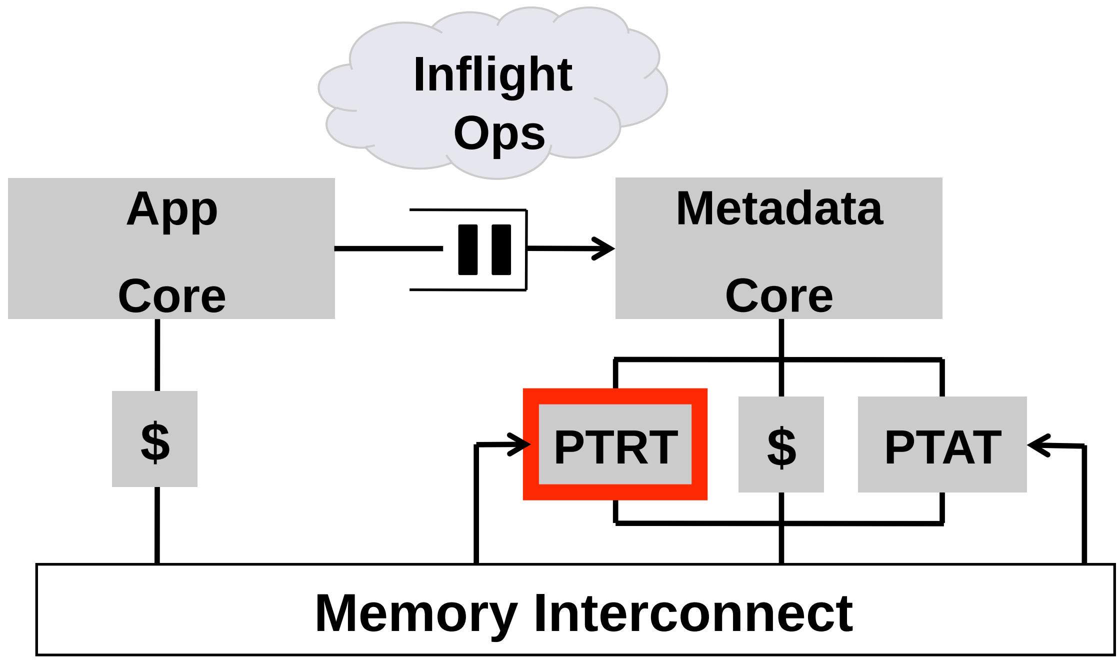
- Keep track of data coherence requests
  - Provides log of *memory races* between threads
- Enforce same ordering on metadata
  - Core A requests data from Core B ➔ Tag Core A requests metadata from Tag Core B
  - Intervening accesses delayed for consistency
- Ensures atomic view of (*data, metadata*)
  - Replaying memory ordering ensures consistency

Consistency Mechanism



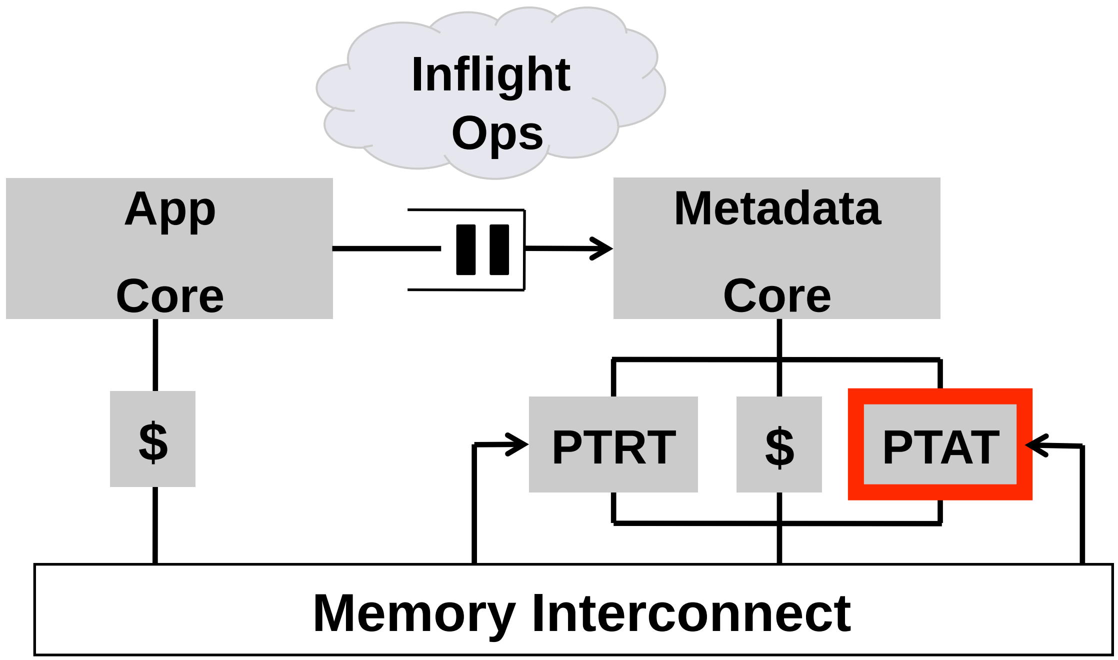
- Every instruction associated with unique ID
- Inflight Operations
  - Maintains information about the instruction in flight
  - Similar to decoupling queue for DIFT coprocessor

Consistency Mechanism



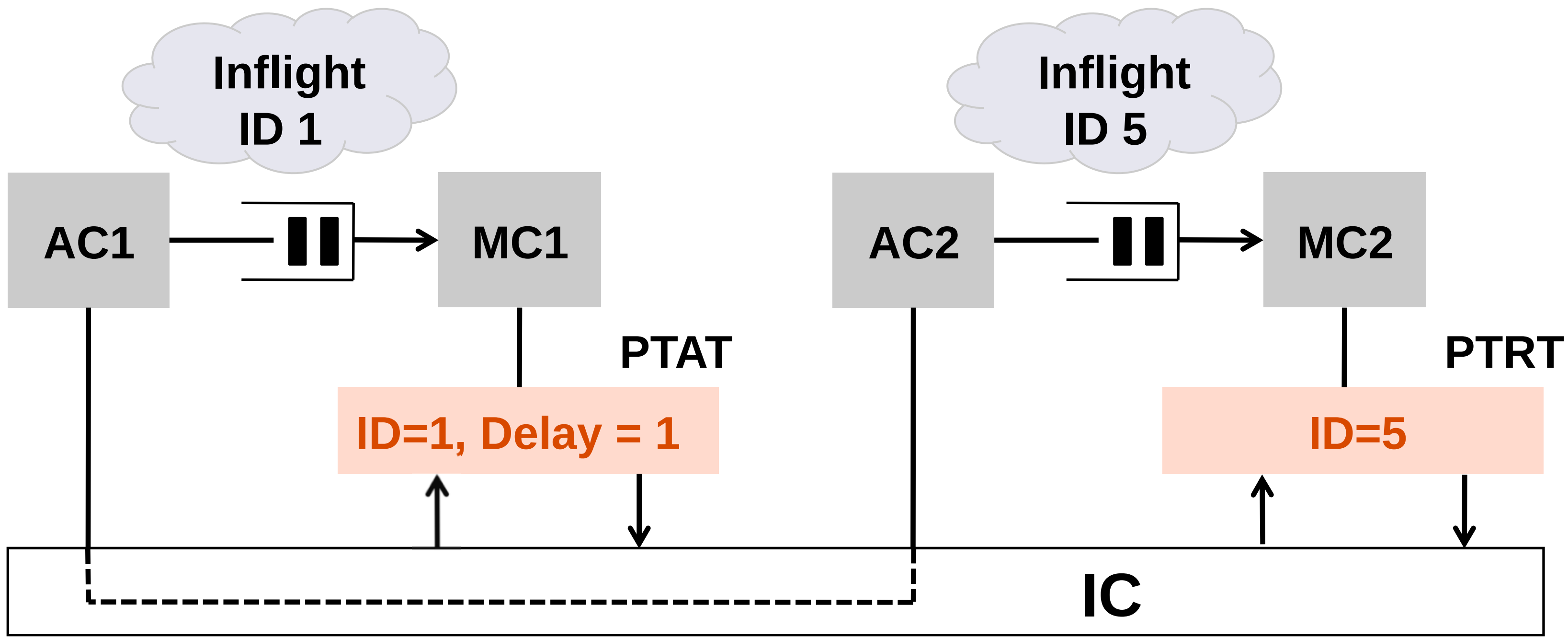
- PTRT = Pending Tag Request Table
- Logs app core's coherence requests
- Metadata core indexes PTRT by instruction ID
  - Directs metadata request to associated core

Consistency Mechanism



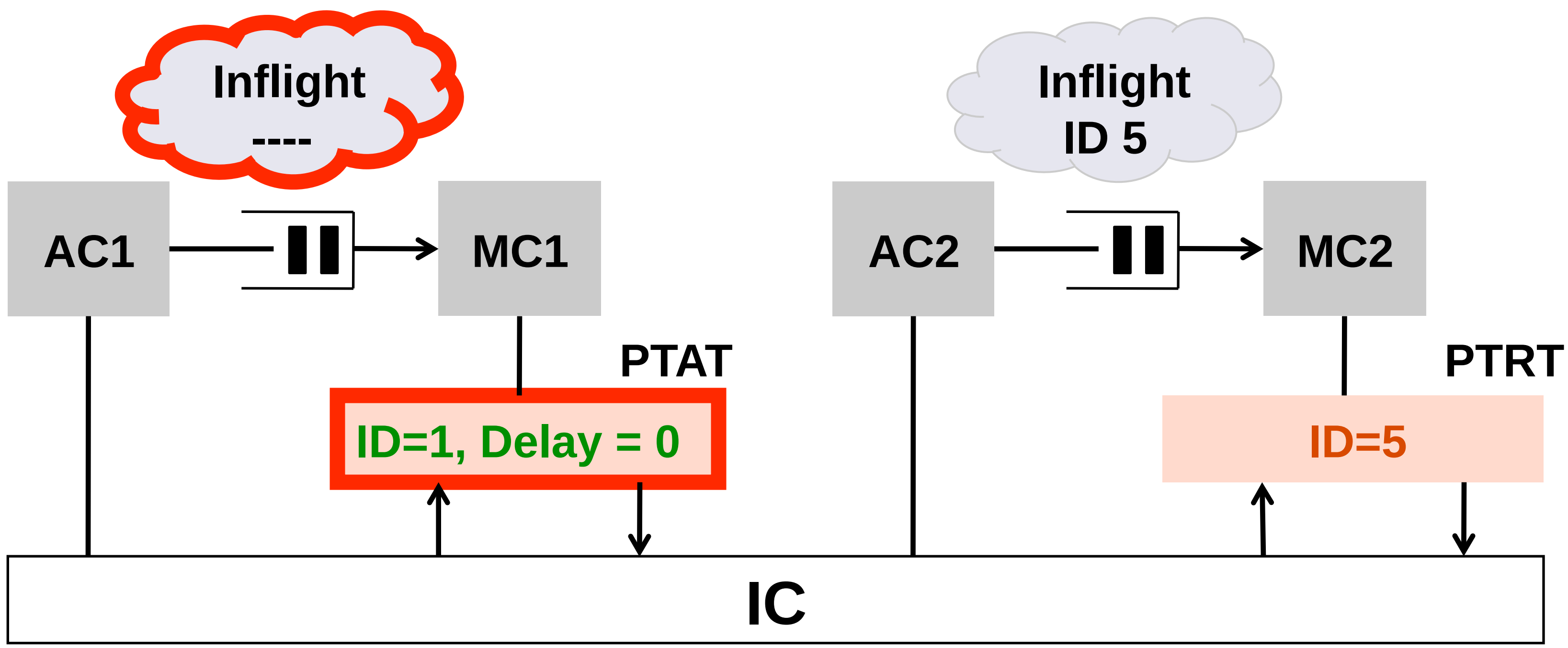
- PTAT = Pending Tag Acknowledgement Table
- Logs last instruction ID to update data value
- On corresponding metadata request
  - Check if insn tag processing complete before replying

Consistency Protocol



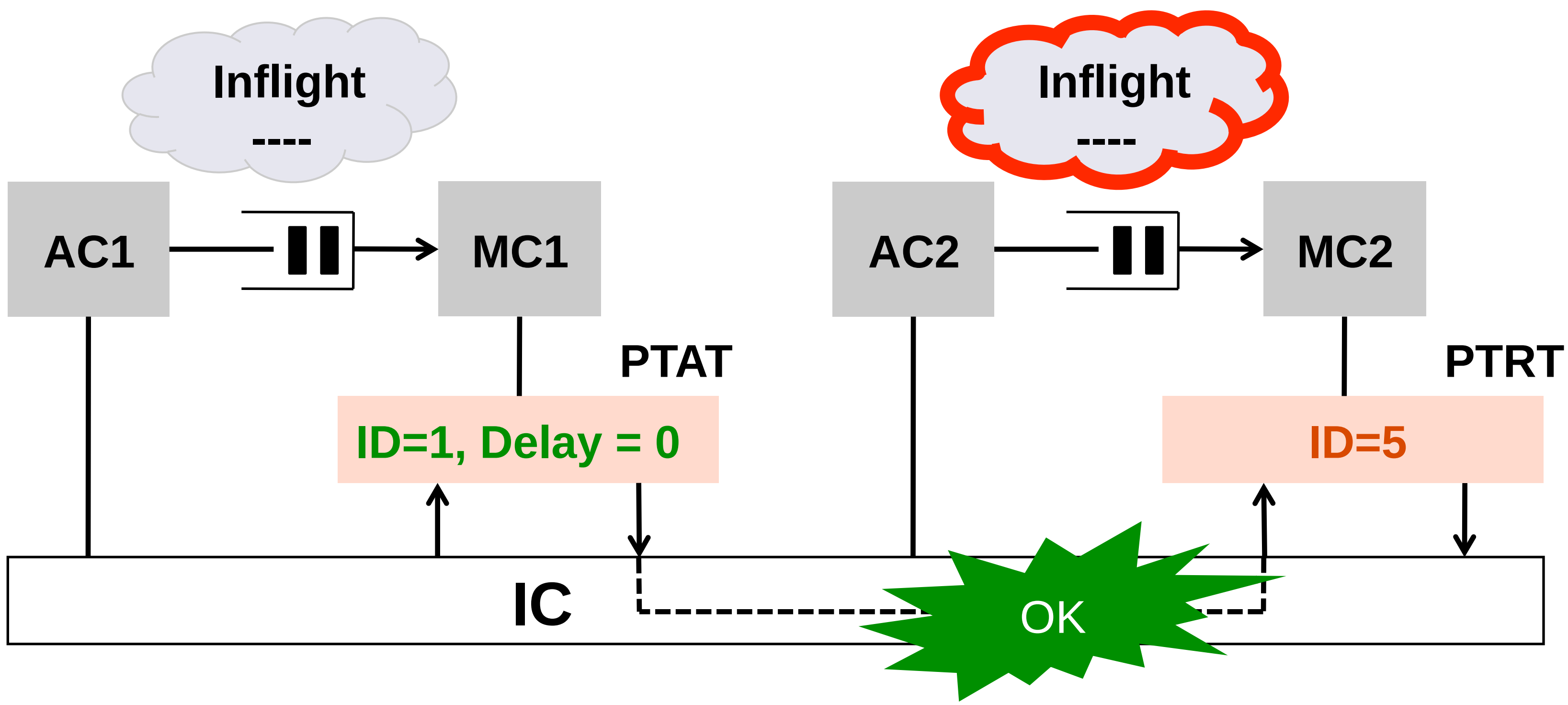
(a) Update PTAT of responder and PTRT of requestor

Consistency Protocol



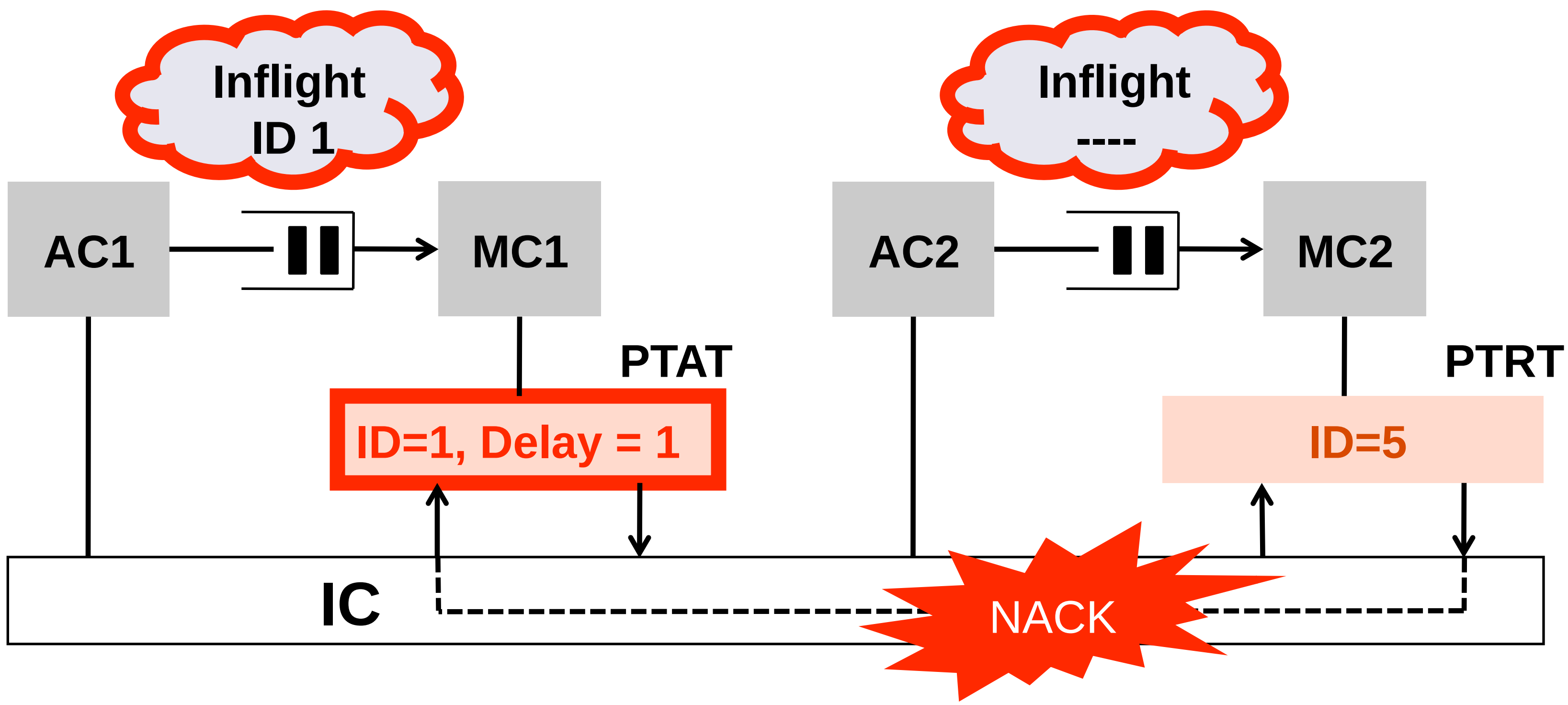
(b) Reset delay bit in PTAT of responder

Consistency Protocol



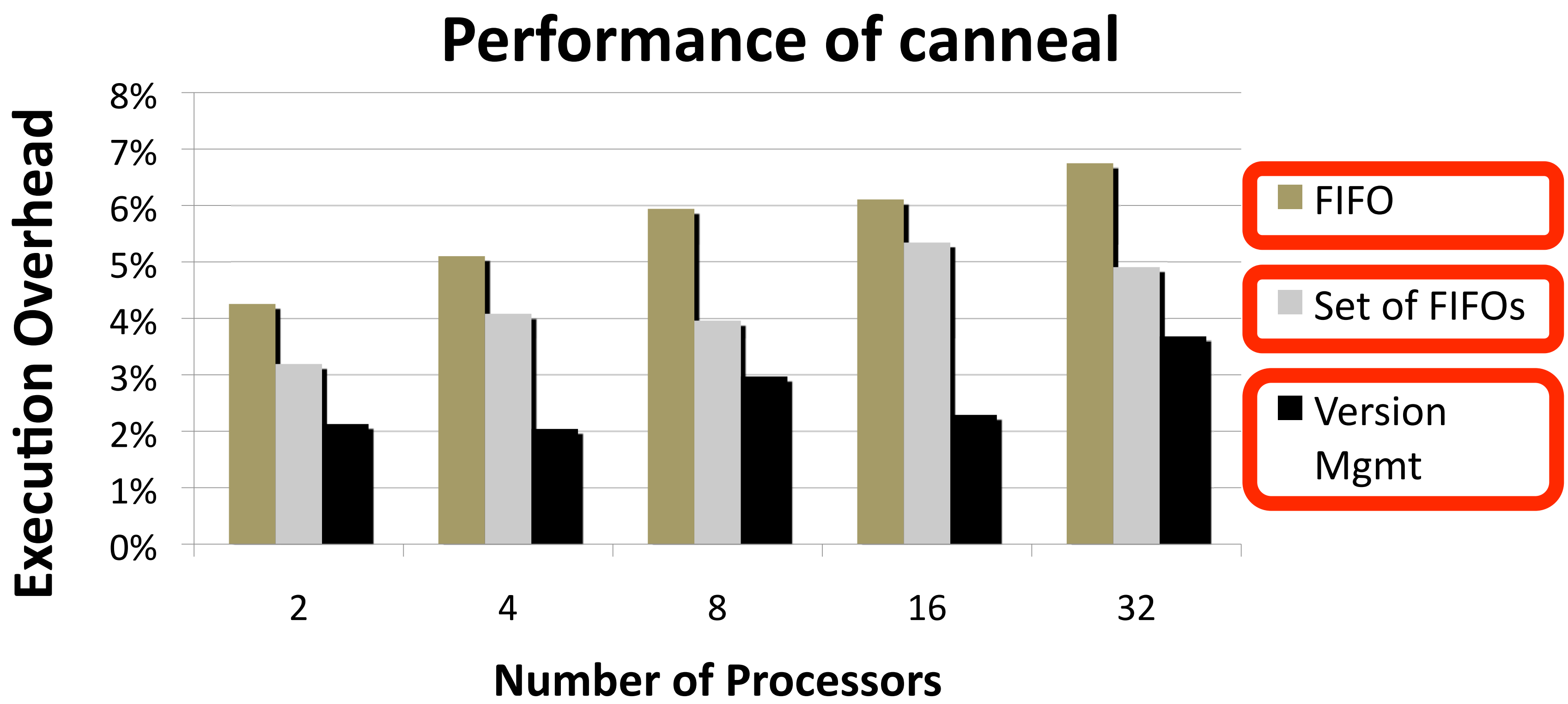
(c) Issue metadata request, receive response

Consistency Protocol



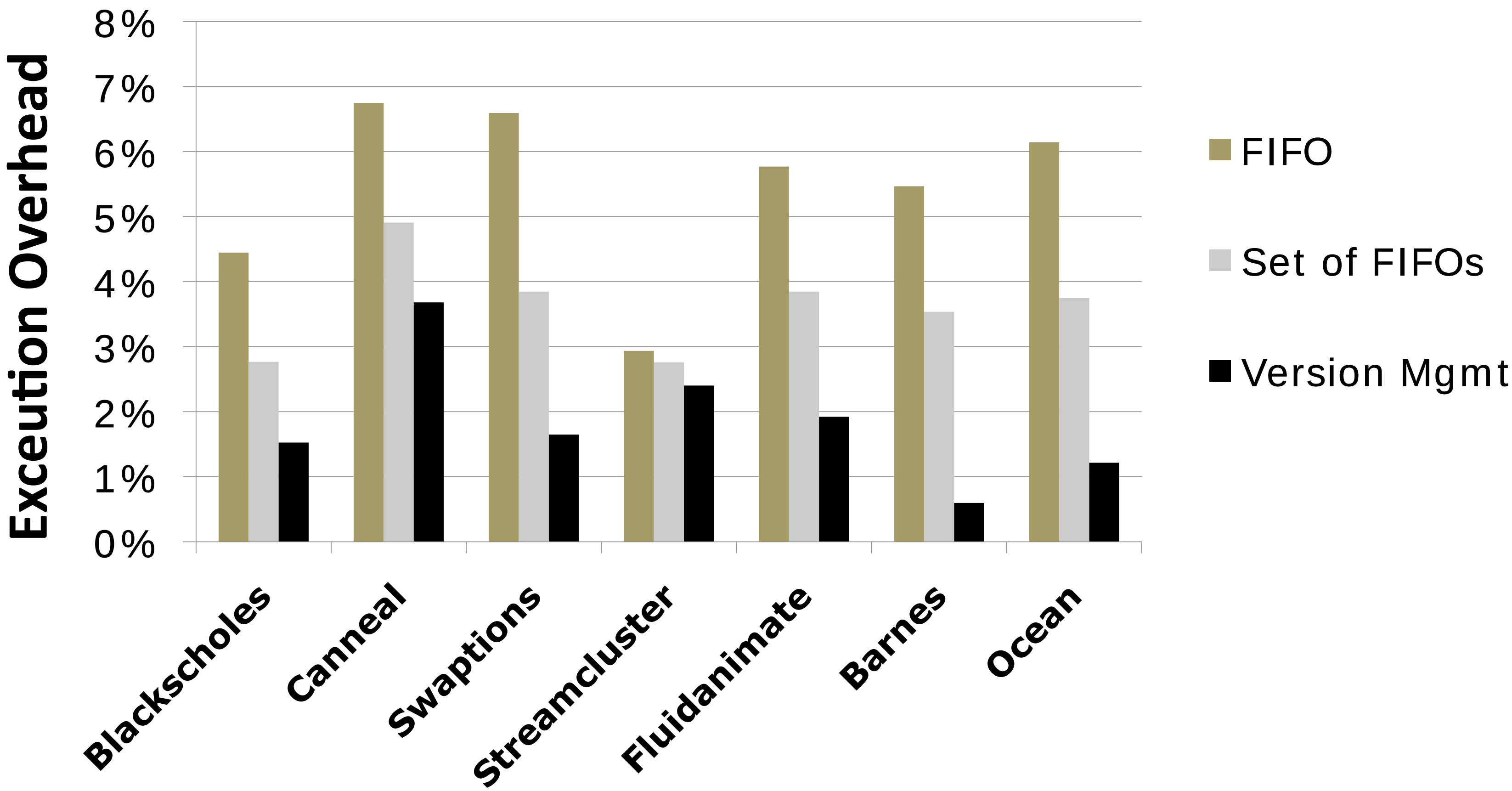
(d) Early metadata request NACKed

System Performance Overheads



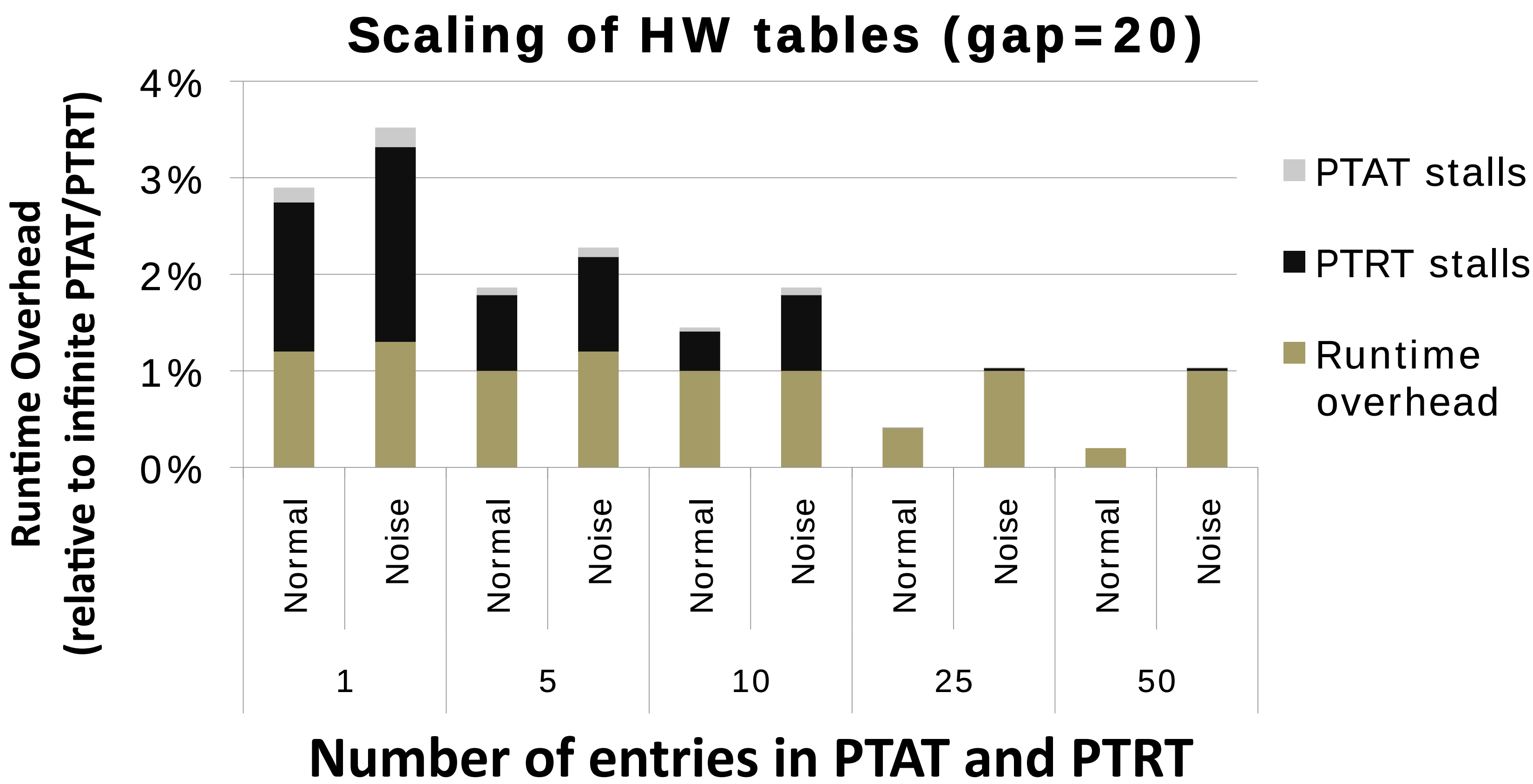
- Different configurations for PTAT:
  - [illegible]

Worst-case Overheads



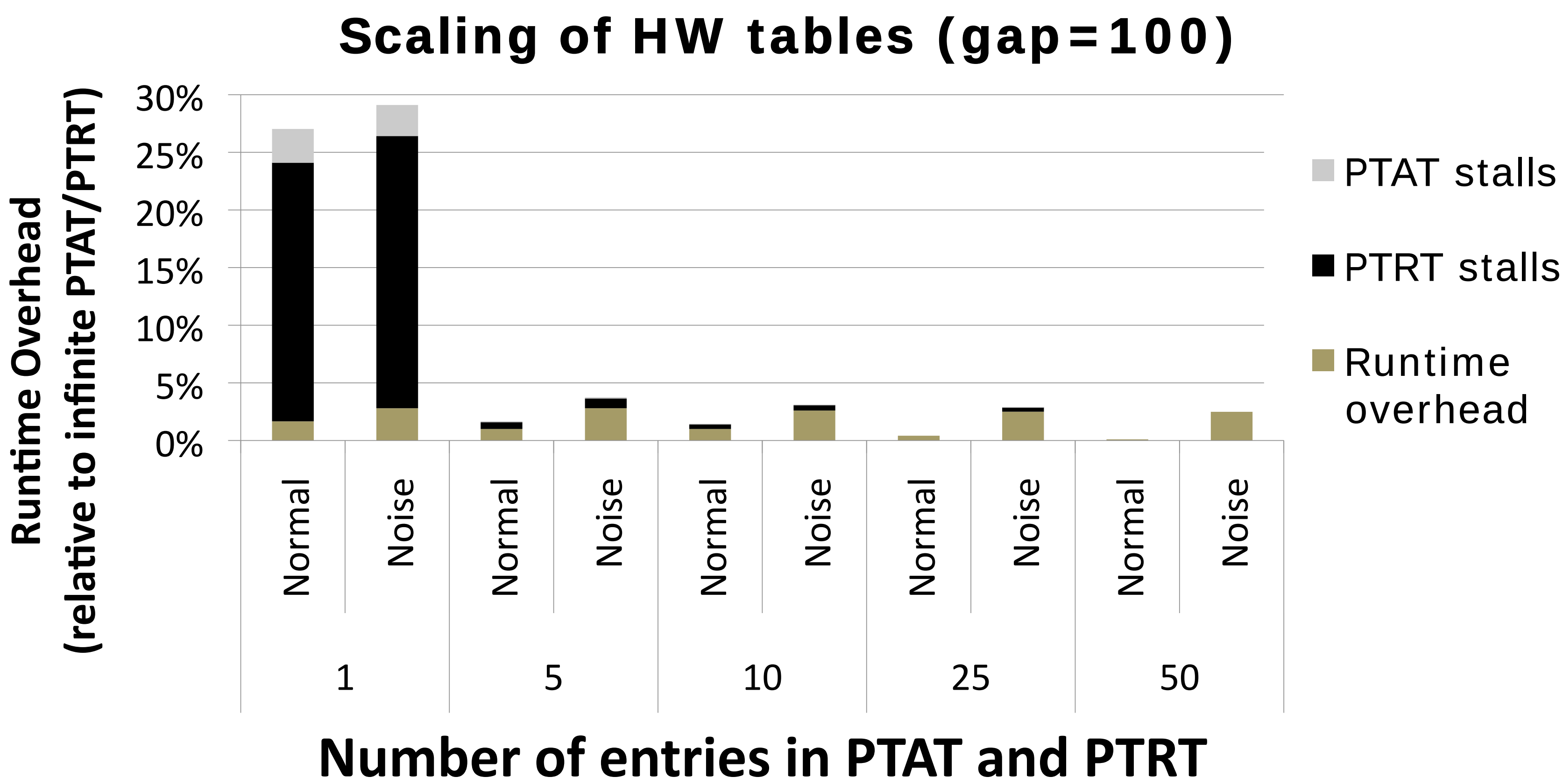
- Performance overheads **< 7%** with 32 processors
- Even simple FIFO design has good performance

Scaling the Hardware Tables



- Worst-case lock contention micro-benchmark
  - Simulates the coprocessor environment

Scaling the Hardware Tables



- Worst-case lock contention micro-benchmark
  - Simulates the log-based architecture environment

Outline

- DIFT overview
- Raksha: hardware support for DIFT [WDDD'06, ISCA'07]
  - Flexible HW design for efficient, practical DIFT on binaries
- Decoupling DIFT from the processor [DSN'09]
  - Using a coprocessor to minimize changes to the main core
- Multi-processor DIFT [MICRO'09]
  - Ensuring consistency between data and metadata under decoupling
- Loki: hardware support for information flow control [OSDI'08]
  - Enforcement of app security policies with minimal trusted code

Dynamic Information Flow Control

- Single abstraction across all system layers
  - Security policies as restrictions on data movement
- Basic idea
  - Every object is marked with a label
  - On accesses, look up label to get a R/W/X permission
- Building upon flow control
  - App policy expressed using labels directly
  - Labels describe protection domains with flexible sharing

Loki: HW Support for Info Control

- Loki implements tagged memory
  - Each word of physical memory associated with a 32-bit tag
  - Tags map to access permissions (R/W/X) for protection domain
  - **Fine-grained** access control
- **Simplifies** security enforcement
  - SW manages tags, but HW enforces security policies
  - Helps maintain security in face of compromised OS
- Ties security policies to **physical** resources
  - Physical resource policies avoid ambiguity
- Allows for a smaller TCB
  - Reduced the TCB of HiStar by over a factor of two

Conclusion

- Hardware DIFT is a promising security solution
  - Prevents HL/LL attacks, is fast, does not need src code
- Co-developed Raksha, a **flexible** hardware design for **efficient, practical** DIFT on binaries
  - DIFT coprocessor to minimize changes to main core/cache
  - Mechanism for safe DIFT on multithreaded binaries
  - Including **real** full-system prototypes (HW+SW)
- Extended hardware DIFT techniques to implement **information flow control**
  - Allows for significant reduction in size of OS' TCB

Bibliography

- "**Deconstructing Hardware Architectures for Security**," Michael Dalton, Hari Kannan, Christos Kozyrakis. *5th Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD) at ISCA*, Boston, MA, June 2006.

- "**Raksha: A Flexible Information Flow Architecture for Software Security**," Michael Dalton, Hari Kannan, Christos Kozyrakis. *Proceedings of the 34th Intl. Symposium on Computer Architecture (ISCA),* San Diego, CA, June 2007.

- "**Raksha: A Flexible Architecture for Software Security**," Hari Kannan, Michael Dalton, Christos Kozyrakis. *Technical Record of the 19th Hot Chips Symposium,* Palo Alto, CA, August 2007.

- "**Thread-Safe Dynamic Binary Translation Using Transactional Memory**," JaeWoong Chung, Michael Dalton, Hari Kannan, Christos Kozyrakis. *Proceedings of the 14th Intl. Symposium on High-Performance Computer Architecture (HPCA)*, Salt Lake City, UT, February 2008.

Bibliography cont'd

- "**Real-World Buffer Overflow Protection for Userspace and Kernelspace**," Michael Dalton, Hari Kannan, Christos Kozyrakis. *Proceedings of the 17th Usenix Security Symposium,* San Jose, CA, July 2008.

- "**Hardware Enforcement of Application Security Policies**," Nickolai Zeldovich, Hari Kannan, Michael Dalton, Christos Kozyrakis. *Proceedings of the 8th Usenix Symposium on Operating Systems Design & Implementation (OSDI)*, San Diego, CA, December 2008.

- "**Decoupling Dynamic Information Flow Tracking with a Dedicated Coprocessor**," Hari Kannan, Michael Dalton, Christos Kozyrakis. *Proceedings of the 39th Intl. Conference on Dependable Systems and Networks (DSN)*, Estoril, Portugal, June 2009.

- "**Ordering Decoupled Metadata Accesses in Multiprocessors**," Hari Kannan, *Proceedings of the 42nd Intl. Symposium on Microarchitecture (MICRO),* New York City, NY, December 2009.